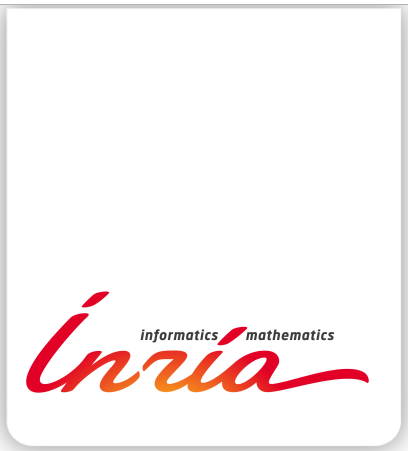
Verification for Security Monitoring SLAs in IaaS Clouds: the Example of a Network IDS

Amir Teshome Wonjiga, Louis Rilling, Christine Morin

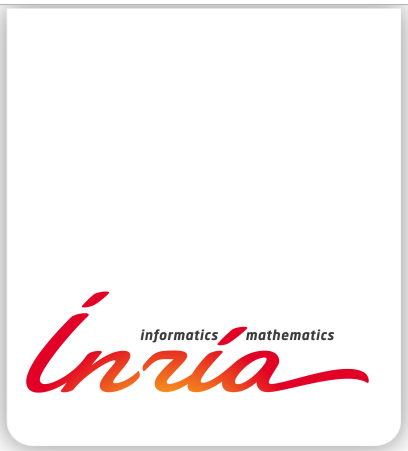


RESEARCH REPORT

N° 9091

September 2017

Project-Teams : Myriads



Verification for Security Monitoring SLAs in IaaS Clouds: the Example of a Network IDS

Amir Teshome *, Louis Rilling †, Christine Morin *

Project-Teams : Myriads

Research Report n° 9091 — September 2017 — 17 pages

Abstract: In an IaaS cloud the physical infrastructure is controlled by service providers, including its security monitoring aspect. Clients hosting their information system need to trust and rely on what the providers claim. At the same time providers try to give assurance for some aspects of the infrastructure (e.g. availability) through service level agreements (SLAs). We aim at extending SLAs to include security monitoring terms. In this paper we describe the challenges to reach this goal, we propose a three-steps incremental strategy and we apply the first step of this strategy on the case of network IDS (NIDS) monitoring probes. In this case study we select a relevant metric to describe the performance of an NIDS, that is the metric can figure in an SLA and can be measured to verify that the SLA is respected. In particular we propose an in situ verification method of such a metric on a production NIDS and evaluate experimentally and analytically the proposed method.

Key-words: Service Level Agreements (SLAs), Network IDS (NIDS), IDS evaluation metrics, security monitoring, SLA verification

* Inria
† DGA

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Vérification pour les SLA de supervision de sécurité dans les Clouds de type IaaS: exemple d'un IDS réseau

Résumé : Dans un cloud de type IaaS l'infrastructure physique est contrôlée par les fournisseurs de service, notamment pour l'aspect supervision de la sécurité. Les clients qui y font héberger leur système d'information doivent faire confiance et se reposer sur les affirmations des fournisseurs. En même temps les fournisseurs essaient de donner des garanties sur certains aspects de l'infrastructure (par exemple la disponibilité) à l'aide de contrats de niveau de service (ou *service-level agreements* – SLA). Notre objectif est d'étendre les SLA pour y inclure des clauses sur la supervision de sécurité. Dans cet article nous décrivons les défis relatifs à cet objectif, nous proposons une stratégie incrémentale en trois étapes et nous appliquons la première étape de cette stratégie au cas des sondes IDS réseau (NIDS). Dans l'étude de ce cas nous sélectionnons une métrique pertinente pour décrire la performance d'un NIDS, c'est-à-dire une métrique pouvant figurer dans un SLA et pouvant être mesurée pour vérifier que le SLA est respecté. En particulier nous proposons une méthode de vérification *in situ* d'une telle métrique sur un NIDS en production et nous évaluons expérimentalement et analytiquement la méthode proposée.

Mots-clés : Service Level Agreements (SLA), IDS réseau (NIDS), métriques d'évaluation des IDS, supervision de sécurité, vérification de SLA

1 Introduction

Before the introduction of cloud computing, organizations used to host their own computing resources (networks, servers, storage, applications, and services). Using clouds, organizations (called tenants) benefit from cost reduction (in both building and management), an elastic infrastructure and broad network access, but they also face new problems in terms of security and lack of open standards.

One of the risks of moving to a cloud is losing full control of the information system infrastructure. The service provider is in charge of monitoring the physical infrastructure and providing the required service to tenants. This pushes tenants to trust providers.

Service providers give assurance on some aspects of the service but, as of today, *not in security monitoring*. In our work, we aim to allow providers to provide tenants with such security monitoring guarantees.

Security Monitoring is the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions [1]. By monitoring a system it is possible to detect suspicious behaviors and react before severe damage. Intrusion Detection Systems (IDS) and logs from firewalls are often used for this purpose. In particular Network IDSs (NIDS) monitor network traffic for suspicious activity and generate alerts when potentially hostile traffic is detected [2].

A Service Level Agreement (SLA) is a contract between a tenant and the service provider. SLAs describe the provided service, the rights and obligations of both parties and state penalties for when the specified terms are not respected. Such an SLA is composed of Service Level Objectives (SLO), which describe the targeted quality of service using Key Performance Indicators (KPI). Hence, SLAs help providers to build more trust.

SLAs differ between service providers. The SLOs also differ by service models (SaaS, PaaS, IaaS) and deployment models (Public, Private, Hybrid). Although the verification method proposed in this paper could be used in all deployment models, we focus on the case of SLAs in Infrastructure as a Service (IaaS) clouds.

One of the core concepts of IaaS clouds is sharing a single physical machine for multiple users through virtualization. We believe that in such an environment the responsibility of monitoring the security can not be left to either the provider or tenants alone. Service providers can not perform full security monitoring without having some knowledge of the services running in tenants environments. Also security monitoring only from tenants side misses visibility on the underlying physical infrastructure as they do not have control on it.

We believe that mutual responsibility, where providers give guaranteed security monitoring for service lists disclosed by tenants, can be achieved through SLAs. The trade-off between tenant's private information disclosed and the monitoring service offered should also be studied. In our work we assume that each tenant needs to provide the set of services to be monitored. Based on this a provider will offer SLA terms which specify the performance of monitoring services with clear metrics.

To achieve including security monitoring terms into SLAs, our approach consists in dividing the problem into sub-problems according to the SLA life cycle. Then we focus on one of the phases of the SLA life cycle called *Verification*, in the case of NIDSs.

In this paper we present two contributions. First to decompose the problem of including security monitoring terms in IaaS cloud SLAs, we propose a three-steps incremental strategy based on the SLA life cycle. Second to apply the first step of this strategy on the case of NIDS probes, we adapt an NIDS evaluation method using a new attack injection algorithm, to implement the Verification phase of the SLA life cycle. We evaluate experimentally and analytically this verification method to show its feasibility in an IaaS production environment from the viewpoint of performance, usefulness and security.

In the rest of this paper, Section 2 presents related work, Section 3 details the problem and challenges to be addressed, Section 4 describes our approach for SLA verification on the specific case of NIDSs, Section 5 presents an evaluation of the proposed method and Section 6 concludes the paper.

2 Related Work

In this section we present related work in three parts. First we briefly present Cloud SLAs and the closest approaches to define SLAs for security monitoring in clouds. In the next two parts we focus on NIDSs since the case of these monitoring probes is studied more specifically in this paper. We detail metrics that were defined to measure the quality of service of NIDSs, and then we present NIDS verification methods.

2.1 Cloud SLA and Security Monitoring

As an example of cloud SLA, Amazon cloud service (EC2) [3] offers an availability of more than 99.95%, and gives 10% service credit in return if this is not respected. This service credit becomes 30% if the availability is less than 99.0%. There exists some works on both creating security monitoring devices for a cloud [4] and defining languages and frameworks for SLA description [5]. The domain specific language proposed in [4] describes the detection algorithms of the IDS rather than Service-Level Objectives (SLO), for example a set of rules that can be negotiated before figuring in an SLA. In other words, the language is too low-level to describe SLOs.

The work presented in [6] and works under SPECS project [7] address a related problem. They describe a framework to manage security SLA life cycle (described in Section 3). Aside from the difference between guaranteeing security in cloud and assuring security monitoring in cloud, the SLO used in their work fails to describe the effectiveness of security devices.

To our knowledge, no previous work[1] defines security monitoring terms for SLAs. The difficulty is that SLA terms related to security monitoring need to be verifiable in the cloud setup. In this paper we propose a verification method for cloud security monitoring SLA terms concerning NIDSs.

2.2 Evaluation Metrics

We want to measure the effectiveness of an NIDS. The basic metrics include *True Positive* (TP) and *False Positive* (FP) (resp. *True Negative* (TN) and *False Negative* (FN)). From these basic metrics the True Positive Rate (or detection rate) and False Positive Rate are calculated as $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$ respectively.

Receiver Operating Characteristic (ROC) curves [9] correlate the two complementary metrics (TPR and FPR) and are used to graphically compare IDSs. But all the three metrics (TPR, FPR, and ROC) are impacted by the base rate fallacy problem [10].

Base Rate: The base rate fallacy – which consists in neglecting prior probabilities when judging the probabilities of events – is studied in different fields of decision making [11, 12]. For NIDSs the base rate is the probability $B = P(I)$ that a packet is part of an intrusion. Axelsson [10] described its effect on measuring the performance of IDSs.

Studies on IDS evaluation metrics found that in realistic production sites the base rate is close to zero. Gu *et al* [13] assumed base rates in the range of $\{1 \times 10^{-2} - 1 \times 10^{-6}\}$. In 1999 Axelsson [10] supposed that the maximum value was $\{2 \times 10^{-5}\}$ (2 intrusions per day from

[1] The preliminary idea of this work was presented in [8].

$1,000,000$ records, an intrusion affecting 10 records in average). In our experimental evaluation we use a base rate in the range of $\{1 \times 10^{-2} - 1 \times 10^{-3}\}$.

Positive Predictive Value (PPV) and Negative Predictive Value (NPV): $PPV = P(I \mid A)$ and $NPV = P(\neg I \mid \neg A)$ take the base rate into account but they are still two complementary metrics to measure the effectiveness of an IDS.

Intrusion Detection Capability (C_{ID}): Gu *et al* introduced the C_{ID} metric in [13]. Let X be the random variable representing the IDS input as either "attack" or "legitimate" packets and Y the random variable representing the IDS output where a packet can be detected as intrusive or non-intrusive.

- **Note**: *base rate* $B = P(X =$ "attack")
- Let $H(X)$ be the entropy of X as defined in the information theory.
- Let $I(X;Y)$ be the mutual information which measures the amount of information shared between the two random variables.
- The Intrusion Detection Capability (C_{ID}) is defined as: $C_{ID} = \frac{I(X;Y)}{H(X)}$

The value of C_{ID} ranges in $[0, 1]$ and increases with the IDS ability in accurately classifying the input packets. This unified metric takes the base rate and other metrics into account.

2.3 Security Monitoring Setup Evaluation

To measure the effectiveness of an NIDS in a cloud environment, Probst *et al.* [14] describe a method in two phases: an analysis of network access control followed by the IDS evaluation in a cloned infrastructure based on the set of services running in the virtual infrastructure. Before this work, Massicotte *et al.* [15] used a virtual infrastructure to generate traffic traces and used the trace to evaluate IDSs hosted in physical servers. Both approaches inject attack traffic to measure the effectiveness of an NIDS, the former in a cloned given virtualized infrastructure and the latter as a generic product. However neither of them considers injecting a mix of realistic proportions of legitimate and attack traffic, which leads to the *base rate fallacy* issue (see Section 2.2).

In this paper, to measure the effectiveness of an NIDS in an IaaS cloud, we keep the idea of injecting attack traffic and improve the relevance of the obtained measures by two means: we do not clone the production NIDS and the injected traffic is a mix of controlled proportions of legitimate and attack traffic.

3 Problem Decomposition

In this section we detail and decompose the problem of defining cloud SLA terms for security monitoring in smaller problems. To guide and validate this decomposition we study the specific case of NIDSs as security monitoring probes.

We identify four main challenges:

1. Virtualized infrastructures are dynamic and malleable, since creation, deletion and migration of VMs are frequent. Security monitoring terms must anticipate such changes. Moreover, the security monitoring process (definition, enforcement and verification of SLA terms – see below) must be automatic.
2. There is no standard to express precise security monitoring properties independently from the devices used.

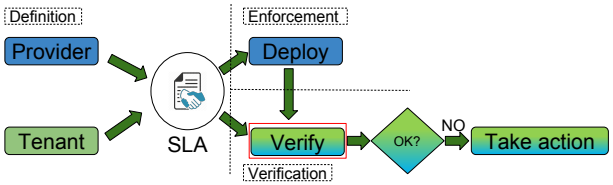


Figure 1: An Overview of SLA Life-Cycle

3. To our knowledge, no method allows to automatically implement an abstract, tenant-defined, security monitoring policy on a set of security monitoring devices, where the policy implementation can be verified afterwards.

4. No methods evaluate full security monitoring setups in clouds.

In this paper we present our work to address Challenges 1 and 4. The other challenges are left for future work.

Current cloud SLA standards define different categorizations of SLA life cycle phases (e.g. SLA-Ready [16] and WSAG [17]). In SLA-Ready the phases are categorized as Acquisition, Operation and Termination phases. In [18] the identified phases are Negotiation, Implementation, Monitoring, Remediation and Renegotiation. If we set aside differences in terminology, we can summarize the main phases in an SLA life cycle as *SLA definition*, *Enforcement* and *Verification* (see Figure 1). Each of these phases could be split in smaller granularity as described in [19].

SLA Definition is a pre-negotiation phase where service providers draft SLA templates according to the services they provide. They should specify the SLOs with clear KPIs. Once the SLA is defined and agreed it should be enforced in the requested infrastructure. The service provider is responsible for deploying the requested infrastructure while respecting the SLA and towards achieving SLOs in the agreement.

After deployment the infrastructure should be monitored regularly to verify its compliance with SLOs promised in the SLA. This step can be performed either by providers or tenants. Transparency is important if providers do the verification. When an SLA is violated tenants have to react to initiate the next step (remediation, renegotiation...). When a tenant reports a violation of SLA, providers have the option to do the verification and check the report before proceeding to the next phase.

As a practical example, we present the SLA life cycle of the *availability* SLA terms offered by Amazon Web Services [3]. The definition of these terms is as follows:

"AWS will use commercially reasonable efforts to make Amazon EC2 and Amazon EBS each available with a Monthly Uptime Percentage of at least 99.95%, in each case during any monthly billing cycle (the 'Service Commitment'). In the event Amazon EC2 or Amazon EBS does not meet the Service Commitment, you will be eligible to receive a Service Credit as described in Table 1."

Table 1: Amazon Service Commitments and Service Credits

Monthly Uptime Percentage	**Service Credit Percentage**
Less than 99.95% but >= 99.0%	10%
Less than 99.0%	30%

Note that this is the SLA subsection in which the SLOs were described. The full SLA template clarifies every term in an unambiguous way and lists the cases where there are SLA exclusions.

The *enforcement* of this SLA term is accomplished by deploying the services (i.e. instantiating the requested VMs) at the appropriate time and on the requested location. After the service is deployed *verifications* can be done by both parties (provider and tenant) and in the event of SLA violation by the provider, tenants can claim the reward stated in the agreement.

To address the three phases of the SLA life-cycle for the case of security monitoring, we follow an incremental approach that starts by choosing appropriate metrics (KPIs) to define SLOs. In the case of an NIDS, the KPIs should in particular quantify the NIDS effectiveness, that is its capability to detect harmful attacks. After choosing appropriate KPIs, we design verification mechanisms for security monitoring setups (see Section 4 for the case of an NIDS). This will give us insights in the expected efficiency and effectiveness of different strategies to setup security monitoring. From these insights, we should derive heuristics for automatically computing security monitoring setups out of SLA terms.

4 An SLO Verification Method for NIDSs

In this section we propose a verification method that an IaaS cloud tenant can use to verify that SLOs for an NIDS are reached. We present first the threat model assumed, second the class of SLOs considered, third an architectural view of our method, and fourth how we compute the KPIs of the SLOs.

4.1 Threat Model

We do not take into account hardware attacks. The cloud provider is assumed honest and the provider infrastructure is assumed not compromised. Thus attacks can only originate from tenant input to the cloud API, from virtual machines (VMs), and from outside the cloud infrastructure.

In particular no low-level network attacks can reach the virtual switches and change their behavior, since traffic from outside the cloud arrives as IP packets that are routed towards the VMs by an edge router, and traffic from VMs is encapsulated in virtual LANs.

It is important to note that the negotiation of SLAs will cover only known types of attacks. Indeed, service providers will not commit for unknown types of attacks. However the list of monitored attacks can be regularly updated in order to include newly discovered vulnerabilities and attacks.

Note that although the provider is assumed honest, SLAs need to be verified because (i) by definition SLAs must be verifiable and (ii) an SLA violation may be unintended and result from provider-implemented heuristics being imperfect.

4.2 Class of SLOs Considered

In the SLA definition phase, the provider and tenants set lists of services to be monitored, lists of known attacks (normal behaviors of production traffic in the case of anomaly-based IDSs) and the expected effectiveness of NIDSs described with the C_{ID} metric. These lists of attacks are the intersections of users requirements and security monitoring services proposed by the provider.

The tenants' infrastructures are monitored against those attacks. Attacks are listed with details about the type of tenant service they are targeting, as well as the software and its version implementing the tenant service. Only these attacks can be used for verification purposes.

We choose the C_{ID} metric as KPI to describe the effectiveness of an NIDS in SLAs. C_{ID} not only takes the *base rate* into account but also supersedes other commonly used metrics. Note that our method is also valid for other metrics derived from *TP*, *FP*, *TN* and *FN*.

Table 2: Example of services list and attack types figuring in an SLO

Application	**Version**	**Attacks**
Apache	Apache/2.4.7 (Ubuntu)	Denial of Service, port scan
Mysql	14.14 Distrib 5.6.31	Brute force access
WordPress(WP) (Base for other apps)	V. 4.4.5	None
Instalinker (WP plugin)	V. 1.1.1	XSS
Custom Contact Forms	V. 5.1.0.2	SQL injection

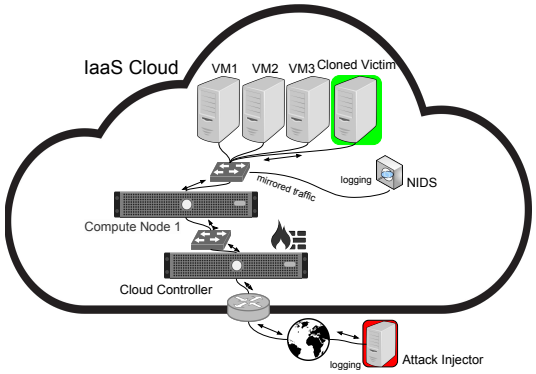


Figure 2: Attack Running Environment

An example of SLO considered is composed of the list of services and attack types presented in Table 2, and a C_{ID} value of at least 0.9 for a base rate $B = 10^{-2}$. The attacks include a Denial of Service and a port scan against the Apache web server, a brute force access to the Mysql database server, a cross site scripting (XSS) attack against the Instalinker WordPress plug-in and an SQL injection against Custom Contact Forms WordPress plug-in. These categories of attacks represent a large section of network attacks [20].

Note that the target C_{ID} value in such an SLO depends on some base rate. In this example the base rate is too high to be realistic, but is suitable for verification purposes (see Section 5.2).

Finally, note that the formal definition of such security monitoring SLOs which includes choosing an appropriate, verifiable C_{ID} value to obtain an effective, real C_{ID} value (with a realistic base rate) is planned for future work.

4.3 Architecture

The verification mechanism performs attack campaigns against a given configuration without damaging the production environment. An example of the attack running environment is shown in Figure 2. The production infrastructure is composed of tenant production Virtual Machines (VM1, VM2, and VM3), hosted on cloud compute nodes (physical servers), and connected to virtual switches. An NIDS is connected to a mirroring port of the virtual switches and analyzes all packets passing through the virtual switches.

We extend this infrastructure with three components:

1. an *Attack Injector*, that injects traffic, containing attacks, to be analyzed by the NIDS;

2. a *Target Virtual Machine* ("Cloned Victim" in Figure 2), to which the injected traffic is redirected, and that exhibits the network behavior of the production VMs;
3. a *Metrics Evaluator* (shown in Figure 3), that given the injected traffic and the NIDS output computes the KPIs used in the SLOs.

4.3.1 Attack Injector

The *Attack Injector* is a physical or virtual machine located inside or outside of the cloud network and it is used to simulate an attacker. The Attack Injector must be able to reach the Target VM. The switch is configured to forward all incoming packets from the Attack Injector to the Target VM. Hence, injected attacks should not affect services running on production VMs.

As shown in Section 2.2, KPI values highly depend on the base rate during the verification phase. However, in real production environments the base rate is likely to be very low and impossible to know. Hence, the base rate used in the verification phase should be based on trade-offs between statistically observed data (see Section 2.2), performance (see Section 5.2), and accuracy when inferring KPI values based on a realistic base rate from verified values.

The novelty of the traffic injection algorithm we propose is to dynamically control the base rate. The algorithm takes three inputs: a set of attacks, a set of legitimate requests, and a target base rate. The base rate is dynamically controlled using the knowledge of the number of packets sent by each attack and each legitimate request. The algorithm runs two loops in parallel that randomly select and inject an attack (resp. a legitimate request) with randomly distributed inter-arrival times. The attacks and legitimate requests inter-arrival time distributions are selected to achieve the target base rate.

4.3.2 Target VM

The *Target VM* is used to simulate the behavior of production VMs (services running in the production VMs). Multiple target VMs could be used in case a single VM is unable to exhibit all the required behaviors. In this SLA verification process we are interested in the network behavior of an application. Any mechanism that can simulate the network behavior of an application could be used in our methodology. In [14] an automaton is used to model network exchanges of an application with a legitimate and a malicious user. More realistic simulations produce more accurate results. One way to perform this is to run the same applications as in the production VMs.

4.3.3 Metrics Evaluator

The packets exchanged between the Attack Injector and the Target VM are logged (dumped) before reaching the NIDS. This log and the output of the NIDS are then used to compute the KPIs.

4.4 KPI Computation

At this stage, we have a predefined set of attacks, the base rate, dumped packets from the communications between the Attack Injector and the Target VM and the output of the NIDS. Using this information we compute the C_{ID} metric of the NIDS. Figure 3 shows the architecture of the Metrics Evaluator. The goals are to see whether the NIDS detected attacks sent from the Attack Injector and to differentiate between true and false positives.

We have to match packets associated to alerts in the output of the NIDS with packets from the communications between the Attack Injector and the Target VM. The output of an NIDS

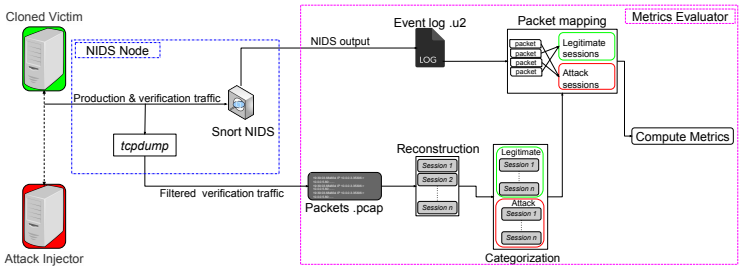


Figure 3: Metric Computation Architecture

is a set of events which are triggered when packets match at least one rule in the NIDS rule set or when monitored traffic exhibit abnormal behavior in the case of anomaly-based NIDSs. An event includes an event ID, source and destination IPs and ports, a time stamp, a set of packets which triggered the event and (if applicable) the rule which generated the event.

The computation starts by reconstructing sessions from the dumped set of packets. Then sessions are categorized into legitimate and attack sessions. This categorization is based on the knowledge about the injected packets. Then packets from the events are mapped to these sessions to get the values of TP and FP. Note that the verification method does not depend on the NIDS software used. Our method can be adapted to different types of NIDS by adapting the parser of the NIDS output. We now detail each metrics computation step.

- *Dumping Packets*: The NIDS is connected to a location where it is able to see every packet passing through the monitored environment. Dumping on such a link allows to see all packets entering in the NIDS. As we only want to reconstruct sessions between the Attack Injector and the Target VM, we use filters to separate those packets.

- *Session Reconstruction*: The output from the previous step is a list of packets. In order to differentiate between legitimate and attack packets from this list we re-assemble the packets in network sessions, which are easier to match with the traffic that is injected.

- *Session Categorization*: Once the packets are assembled into sessions, using our prior knowledge about the injected traffic, we can categorize them into their respective types. To this end, since we know the requests prior to injection, we use some of their characteristic attributes like port numbers used, packet payload and number of packets per session.

- *Packet Mapping*: This is the last step before doing the actual metrics computation. Packets from the NIDS output are mapped to the sessions from the previous step. If a packet matches with an attack session it indicates that the IDS detected that attack. On the other hand if it maps to a legitimate session, the NIDS issued a false positive. From this we can calculate the different metrics described in Section 2.2, including the C_{ID}.

5 Evaluation

In this section we present an evaluation of the SLA verification method for NIDSs proposed in Section 4. The evaluation is composed of an experimental study of the performance impact of the verification method on production VMs, and a correctness, usefulness and security analysis.

5.1 Experimental Setup

We set up an infrastructure as shown in Figure 2 on the Grid'5000 [21] testbed. Each physical machine (node thereafter) featured two Intel Xeon X5570 processors (6M Cache, 2.93 GHz, 4 cores) with 24GB memory. We used OpenStack [22] with one controller and one compute node and Open vSwitch [23] as virtual switch on the compute node. The NIDS is hosted on a third node and is connected to the virtual switches mirroring ports through generic routing encapsulation (GRE) tunnels. The Attack Injector is hosted on a fourth node outside the cloud network.

Three production VMs ran respectively a web server, a database server and a content management server. We added a Target VM that exhibits the behavior of the other VMs by running all three services. All VMs ran Ubuntu server 14.04 with the OpenStack m1.medium flavor (2 VCPUs, 4GB memory and 40GB disk).

Dumping Packets was performed using *tcpdump* on the machine where the NIDS is running. This way we can be sure that both *tcpdump* and the NIDS will see the same level of packet fragmentation. We used filters to dump only the communications between the Attack Injector and the Target VM.

We used *tcpflow* to perform session reconstruction. It reconstructs the data stream and stores each session in a separate file. *tcpflow* understands sequence numbers and correctly reconstructs regardless of retransmissions or out-of-order delivery. However *tcpflow* does not handle IP fragments (the attacks used for this evaluation do not use IP fragmentation).

For this evaluation we used the example SLO presented in Section 4.2. To this end we collected the vulnerable applications and collected and/or developed the attacks listed.

As mentioned in Section 4.3 attacks are injected in parallel with legitimate requests. To this end, we collected a set of legitimate requests: loading a web page, login into Mysql, uploading a file and login into a WordPress account.

Finally, we used the Snort [2] mainstream open source network IDS. We collected or wrote Snort rules for the attacks listed in Table 2.

5.2 Performance Impact

Since our SLA verification method runs in the production network infrastructure but does not involve production VMs, the most expected performance impact is network overhead.

The network overhead is measured as the difference in response time of a given request to a production VM with and without SLA verification running in parallel. To measure this overhead, a first experiment started by simulating production traffic at time t_0. The SLA verification started later at time t_1 and finished at time t_2. The production traffic then continued until time t_3 ($t_0 < t_1 < t_2 < t_3$). This way we can compare the response time in different phases. This experiment was performed using base rate values $B = 10^{-2}$ and $B = 10^{-3}$.

Table 3 shows the average number of attacks and legitimate requests injected over ten rounds using base rate value $B = 10^{-2}$. For this experiment both attack and legitimate request injectors used three processes each to send interlaced verification traffic.

The box plot in Figure 4 shows the mean response time for each type of legitimate request in the three time intervals: "Before" verification, "At verification", and "After" it is finished. The plot shows that the overhead is very small relative to the time needed for a request. We observe a maximum overhead of 6.9% (0.000165 second) in the case of "SQL Login" requests and a minimum of 2.2% (0.001755 second) in the case of "WordPress Login" requests (see Figure 5). For "WordPress Login" requests we can notice with the "After" box plot that the response time increases during the experiment even without verification. Overall in this experiment, the most observable impact of verification seems to be an increased dispersion of response times.

Table 3: Traffic injected using $B = 10^{-2}$ and parallelism degree of 3.
#S = average number of injected flows (attacks or legitimate requests).
#P = average number of packets per flow.

Attacks	**#S**	**#P**	**Legitimate request**	**# S**	**# P**
Denial of Service (DoS)	12	1916	View Blog	2482	6
Port scan	12	1012	WP login	2498	12
Brute force access	12	2000	Mysql login	2479	6
XSS	13	6	Upload file	2501	3111
SQL injection	12	25			

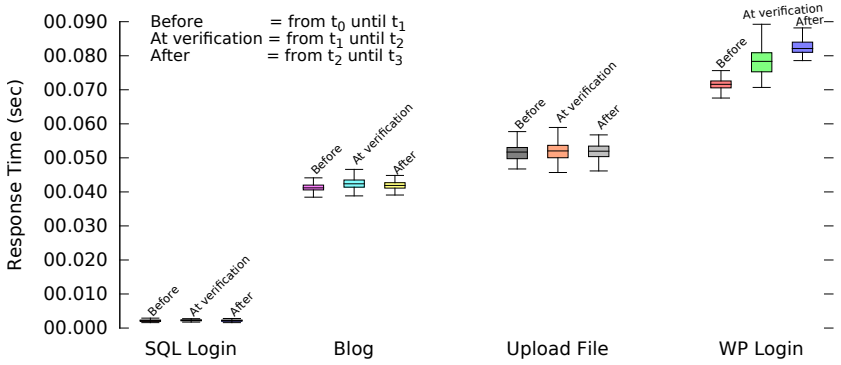


Figure 4: Impact of verification on the mean response time for the four types of legitimate request

Another important factor while performing verification is the base rate. It determines the time required for verification. The experiment to produce the graph shown in Figure 4 took 41.6 minutes using a base rate $B = 10^{-2}$. We also did an experiment with $B = 10^{-3}$, which showed the same overhead but took a much longer time (around 4 hours). This duration is expected because as the base rate decreases, the rate of attack injection declines and it requires a longer time to get enough attack samples to compute accurate statistics.

Decreasing verification time is possible by increasing both the attack and legitimate traffic injection rate but the overhead in the production environment will increase. Figure 5 shows the time needed in further experiments to perform verification using different traffic injection rate. The injection rate can be altered by increasing (decreasing) the number of parallel processes. We performed three experiments using respectively 3, 12 and 30 processes for both attack and legitimate traffic. It took 41.6, 12.5 and 2.45 minutes respectively, while the overhead increases for each case respectively. In other terms there is a trade-off between the time required to perform the verification and the overhead on the production environment.

5.3 Correctness, Usefulness and Security Analysis

In this section, we first show the correctness of our verification process, then its usefulness from tenants and providers perspective and finally we present a detailed security analysis.

5.3.1 Correctness Analysis

To show the correctness of our verification process, we first manually compute the expected C_{ID} value using knowledge from the NIDS configuration, then we run the verification mechanism and check if the result matches the expected value. In the experiment Snort is configured with 49 rules to detect the attacks listed in Table 3. The table also lists the actual number of attacks and legitimate requests injected in our experiment using base rate $B = 10^{-2}$.

We expect Snort to give alerts for almost all true attacks and very few (close to zero) false positives. This results from carefully crafted rules in the experiment setup, which simulates a real production environment. It should be noted that, in a real production environment, achieving such a low false positive rate is difficult. In our experiment we observed an average packet drop rate from Snort of 2.95% over ten rounds. Consequently, we expect false negatives ($FNR \approx 0.03$) as some attack packets are likely not processed by Snort. As a result the expected C_{ID} value should be around 0.95.

Using the data in Table 3, the output of Snort and the packets recorded from the attack campaign, we computed the C_{ID} value as shown in Figure 3. As expected the metrics evaluator outputs $TPR \approx 1.0$ and $FPR \approx 0$, i.e. all processed attacks are detected and no false alarm is generated. We have $FNR \approx 0.07$ and $C_{ID} \approx 0.90$. In our experimentation environment the metrics evaluator thus shows a very small and acceptable difference from the expected FNR value.

5.3.2 Usefulness Analysis

Having such a transparent SLA verification mechanism is useful for both service providers and tenants. Tenants should trust more services they consume as such SLAs guarantee compensation in case of SLO violation. For providers, in addition to increasing their clients trust, providing guaranteed security monitoring service should give economic advantage either directly by providing a paid service model or indirectly by attracting new customers, who were previously not confident enough with the security of clouds.

5.3.3 Security Analysis

In our verification mechanism the Target VM is the only added component intervening in the production environment. The other components (Attack Injector and Metrics Evaluator) could

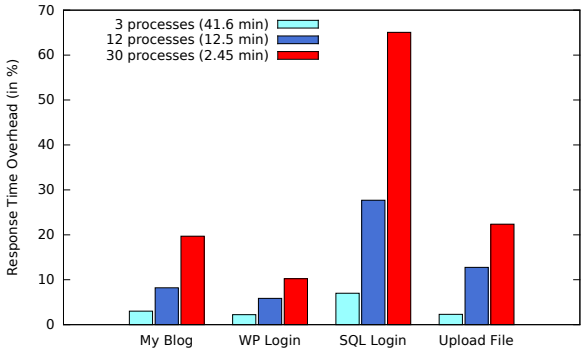


Figure 5: Response time overhead as a result of verification, with $B = 10^{-2}$, the four types of legitimate request and increasing traffic injection rates

be located outside of the production infrastructure. The Target VM is carefully prepared to simulate the behavior of production VMs and is targeted by attacks which are detected by the network IDS, i.e. attacks like cross-VM side-channel [24] (attacks that could affect other VMs in the same host) and undetectable IDS evasions [25] should not be injected in this method.

If an attacker performs IP spoofing types of attacks [26] to send attack packets impersonating verification traffic, this could be dumped as a communication between the Attack Injector and Target VM. This would disrupt session reconstruction and metrics evaluation afterwards. In addition, occurrences of attacks targeting the NIDS [25] during the verification phase could prevent the NIDS from performing as expected and would falsify the final result. To prevent such uncontrollable errors choosing appropriate times for verification could help, for instance during the most idle times of the production environment.

Attack packets injected into the production network are no real threats because the virtual switch is configured to forward all incoming packets from the Attack Injector to the Target VM. This guarantees that attack packets injected to the network do not reach production VMs.

To configure the switch the provider must cooperate since tenants should have no direct control on a shared switch. This cooperation should be incited by the economic advantage brought by providing guaranteed security monitoring.

Packets from the Attack Injector are filtered and dumped, from the same virtual switch mirroring port the NIDS is attached to, for later analysis. In a network where there is very high traffic *tcpdump* may not perform as expected. Packet drops in this phase may affect subsequent steps. In particular if *tcpdump* is writing to disk the speed and disk space should be taken into account. Writing to a disk may decrease the speed and lead to packet loss. Other tools like Moloch [27] could be used to overcome such problems.

We used *tcpflow* to reconstruct sessions from the output of *tcpdump*. But *tcpflow* does not understand IP fragments, that is flows containing IP fragments will not be recorded properly. In cases where the NIDS and *tcpflow* use different IP fragments handling strategy, the reconstruction phase would also be altered. Thus attacks exploiting IP fragmentation could lead to errors in the packet mapping phase. Other tools like Wireshark [28] could be used to overcome this issue.

If there is any external TCP session (communication which is not between the Attack Injector and the Target VM) in the *tcpdump* output, it is reported in the categorization phase. Getting a higher percentage of uncategorized sessions would imply poor filtering and/or reconstruction and it could lead to a false result. In such cases the verifier should observe and decide whether to continue or redo the previous steps.

Finally, as a requirement for security monitoring SLA terms, tenants are required to provide lists of services to be monitored. This disclosure of services should not be a security concern for two reasons. First as we assume the provider is trustworthy, confidentiality of tenants should be respected. Second our methodology does not require knowledge about the actual data processed by the services.

6 Conclusion and Future Work

In this paper we tackled the problem of including security monitoring terms in IaaS cloud SLAs. We presented two contributions. First to decompose the problem we proposed a three-steps incremental strategy based on the SLA life cycle. Second to apply the first step of this strategy on the case of NIDS probes, we adapted an NIDS evaluation method to implement the Verification phase of the SLA life cycle. We evaluated experimentally and analytically this verification method to show its feasibility in an IaaS production environment from the viewpoint of performance, usefulness and security.

As a result first we studied state-of-the-art IDS evaluation metrics and we chose the C_{ID} as a usable KPI in SLOs to describe the effectiveness of an NIDS because it takes the base rate into account and supersedes traditionally used metrics.

Second we proposed an SLO in situ verification method to measure the C_{ID} of an NIDS dynamically using traffic injection. The novelty of our verification method is to improve the relevance of the obtained measures by two means: for the sake of trust we do not clone the production NIDS, and the base rate of the injected traffic is dynamically controlled. The experimental evaluation shows that during verification phases of 40 minutes a reasonable overhead (less than 10%) can be observed on production VMs response times. This overhead could be decreased at the price of longer verification phases.

In future work we will study how to choose verifiable C_{ID} values and add other metrics to describe the NIDS configuration compliance to users' requirements in SLOs. We also plan to extend this approach to other monitoring probes (e.g. firewalls). To complete the approach we will then study heuristics to automatically configure the security monitoring according to SLOs.

References

[1] R. Bejtlich, *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Pearson Education, 2004. [Online]. Available: https://books.google.fr/books?id=O6wBZs7fODIC

[2] M. Roesch, "Snort: Lightweight intrusion detection for networks." in *Proc. LISA '99*, 1999, pp. 229–238.

[3] Amazon, "Making Cloud SLAs readily usable in the EU private sector," Nov. 2016. [Online]. Available: https://aws.amazon.com/ec2/sla/

[4] D. Riquet, G. Grimaud, and M. Hauspie, "DISCUS: A massively distributed IDS architecture using a DSL-based configuration," in *Proc. ISEEE*, 2014.

[5] K. T. Kearney, F. Torelli, and C. Kotsokalis, "SLA: An abstract syntax for Service Level Agreements," in *Proc. GRID*, 2010.

[6] V. Casola, A. De Benedictis, M. Rak, J. Modic, and M. Erascu, "Automatically enforcing security slas in the cloud," *IEEE Transactions on Services Computing*, 2016.

[7] "SPECS," accessed April 2017. [Online]. Available: http://www.specs-project.eu

[8] A. Teshome, L. Rilling, and C. Morin, "Including Security Monitoring in Cloud Service Level Agreements," in *Symposium on Reliable Distributed Systems (SRDS), PhD forum*, Budapest, Hungary, Sep. 2016. [Online]. Available: https://hal.inria.fr/hal-01354975

[9] J. C. Hancock and P. A. Wintz, *Signal detection theory*. McGraw-Hill, 1966.

[10] S. Axelsson, "The base-rate fallacy and its implications for the difficulty of intrusion detection," in *Proc. CCS*, 1999.

[11] R. A. J. Matthews, "Base-rate errors and rain forecasts," *Nature*, vol. 382, p. 766, 1996.

[12] ——, "Decision-theoretic limits on earthquake prediction," *Geophysical Journal International*, vol. 131, no. 3, pp. 526–529, 1997.

[13] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: an information-theoretic approach," in *Proc. ASIACCS*, 2006.

[14] T. Probst, E. Alata, M. Kaâniche, and V. Nicomette, "Automated Evaluation of Network Intrusion Detection Systems in IaaS Clouds," in *Proc. EDCC*, 2015.

[15] F. Massicotte, F. Gagnon, Y. Labiche, L. Briand, and M. Couture, "Automatic evaluation of intrusion detection systems," in *Proc. ACSAC*, 2006.

[16] "SLA-Ready," accessed Nov 2016. [Online]. Available: http://www.sla-ready.eu/

[17] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open Grid Forum*, vol. 128, 2007, p. 216.

[18] A. De Benedictis, M. Rak, M. Turtur, and U. Villano, "Rest-based SLA management for cloud applications," in *2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2015, pp. 93–98.

[19] K. Saravanan and M. Rajaram, "An exploratory study of cloud service level agreements-state of the art review." *KSII TIIS*, vol. 9, no. 3, 2015.

[20] "Mcafee labs threats report 2015," McAfee Labs, Tech. Rep., May 2015.

[21] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[22] "OpenStack," accessed April 2017. [Online]. Available: https://www.openstack.org/

[23] "Open vSwitch," accessed April 2017. [Online]. Available: http://openvswitch.org/

[24] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An exploration of l2 cache covert channels in virtualized environments," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011, pp. 29–40.

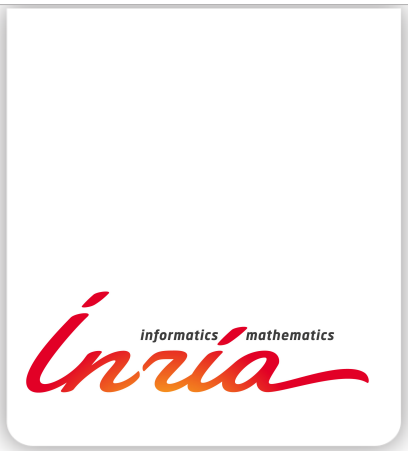
[25] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," DTIC Document, Tech. Rep., 1998.

[26] C. C. Center, "Tcp syn flooding and ip spoofing attacks," *CERT Advisory CA-1996-21*, pp. 1996–2021, 1996.

[27] "Moloch," accessed Nov 2016. [Online]. Available: http://molo.ch/

[28] "Wireshark," accessed Nov 2016. [Online]. Available: https://www.wireshark.org/

Contents



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399